|  |
| --- |
|  |

**Table of Contents**

1.0 Introduction 5

1.1 Executive Summary 5

1.2 Problem Statement 6

1.3 Glossary 6

2.0 Requirements 9

2.1 Server Requirements 9

2.3 Client-Side Requirements 12

3.0 Constraints and Applicable Standards 14

3.1 Economic Constraints 14

3.2 Environmental Constraints 14

3.3 Availability and Support 15

3.4 JavaScript Standards 15

3.6 Third Party Library Standards 18

4.0 Security 22

4.2 Cross Site Scripting (XSS) 25

5.0 Design 28

5.2 Backend: Node.js 29

5.3 File Uploads 30

5.4 CodeMirror 31

5.5 Popcorn.js 31

5.7 UI Mocks 33

5.8 UML Activity Diagrams/Graphics 35

5.9 Use Case 37

6.0 Testing/Data Collection and Analysis Plan 38

6.1 Development Testing 38

6.2 Project Report/Design Testing 39

7.0 Project Management Plan 40

7.1 Scrum Methodology 40

7.2 Timeline 42

8.0 Cost Analysis 50

8.1 Industry Costs 50

8.2 Estimated Development Costs 50

8.3 Required Resources 50

8.4 Employee Salaries and Benefits Cost 52

9.0 Required Independent Learning 53

10.0 Results Section 54

11.0 Discussion Section 55

12.0 Future Work 56

13.0 Appendices 57

13.1 William’s Research: Frontend 57

13.2 Andrew’s Research: File Uploading 62

13.3 Avery’s Research: Core Libraries 65

13.4 Luke’s Research: Backend 67

13.5 Scott’s Research: Database 71

13.6 Team Biographies 74

13.7 Progress Reports 78

13.8 Licenses 80

13.9 API Documentation 81

13.10 FAQ 83

13.11 Additional References 84

13.12 Changelog 84

13.13 Git Commit Log 87

**Introduction**

**Executive Summary:**

PIPSQUEAK is a project with the goal of creating a programming lecture tool that will provide a way for students to learn to program interactively. This will be done via a web page created by the instructor for a specific snippet (or entire file) of code. The instructor will use a standalone application to upload video, audio and code files and add relevant text formatting options (highlight, strikethrough, etc.) synced to specific times in the audio or video file. The idea behind this project is to alleviate or minimize the frustrations when students have trouble understanding specific coding concepts. They can view hands-on, relevant examples with instructor guided videos synced to sensible text formatting options acting on the code.

Additionally, we will allow instructors to easily create the code-learning pages described above by automatically generating a page based on the code sample and any audio/video files the instructor chooses to provide. By making the creation process intuitive, instructors will only have to worry about supplying the necessary files and choosing where to place emphasis in the code. They can do this by adding various effects and/or providing text annotations alongside the code.

The tool we aim to create has the potential to have a tremendous impact on how programming courses are taught and how students learn. Our goal is to enable teachers to effectively teach students how to program and to provide a way for students, who are struggling to grasp programming concepts taught under the current teaching model, to be able to finally understand and eventually master those concepts.

**Problem Statement:**

In the education community there is often a gap between the classroom lectures and student learning. This is particularly true when it comes to designing and writing computer code. Some students may immediately grasp what a segment of code is doing and why it is doing it, while some other students may have difficulty in comprehending even the basics of code. The first group stands to gain a good resource for rapidly learning new concepts and programming languages, but our focus is on the latter group, who stands to reach a whole new level of understanding through our completed project.

The application is designed to enable instructors to teach code in an effective and focused manner. The instructor will be able to dynamically show snippets of code while an audio or video lecture plays of them discussing that particular code snippet. Then the student will be able to focus on a specific portion of code as much as needed to learn and understand what was taught. This should really break the coding process down, significantly lowering the difficulty and stress that are typically associated with mastering a new programming language or skill. As an added advantage, the application will also provide a platform for effective web based coding classes.

**Glossary:**

* PIPSQUEAK – Pending Interlaced Programming and Sound Quality Ensuring Academic Knowledge – the name of our capstone group
* UI – User Interface – interface between user and software
* DOM – Document Object Model – software programming convention for manipulating HTML objects
* CSS – Cascading Style Sheets – programming language for styling web interfaces
* HTML – Hypertext Markup Language – programming language for general website design
* PHP – PHP: Hypertext Preprocessor – server-side programming language
* MVC – Model View Controller – software programming architecture to separate user interface, application control logic and data manipulation
* SQL – Structured Query Language – programming language for handling a relational database
* API – Application Programming Interface – set of protocols, libraries, and other information or tools to make programming easier
* Frontend – Client-side
* Backend – Server-side
* REST – Representational State Transfer – A software architecture that consists of defined architectural constraints
* Node.js – backend programming platform built on Chrome’s JavaScript runtime
* Framework – a software coding package that combines compilers, libraries, APIs and other tools to make software programming in a certain language easier and more efficient for developers
* MongoDB – open source NoSQL database
* NoSQL – Not Only SQL – database that is not a relational database
* AngularJS – frontend framework for developing a JavaScript handled single page web application
* Globals - global functions or variables
* XSS – cross-site scripting – security vulnerability that allows hackers to insert client-side code into web pages
* RDMS – relational database management system
* NPM – node package manager
* TDD – test-driven development
* App – abbreviation for application
* SSL – Secure Sockets Layer
* OS – Operating System
* HTTP – Hypertext Transfer Protocol – primary application protocol used by the web
* RAM – Random Access Memory

**Requirements**

**Server Requirements:**

* Our application will not require a server. It will not even require internet connection. Instead it will be a form of standalone software. The user simply needs to run the PIPSQUEAK application from their own computer, and PIPSQUEAK will take care of the rest.
* We do, however, need to use a server for our development process. In order to use Atlassian Jira, we will need to install it on our own server.
* The Atlassian server will require:
* At least 1 gigabyte of hard drive space.
* At least 2 gigabytes of RAM.
* Java Developers Kit and Java Runtime Environment
* Needs to be an application server.
* A relational database to store issues, which are defined by Atlassian as a software bug, project task, helpdesk ticket, leave request form, etc. Basically issues are the very reason we need to use Jira.

**Client-Side Requirements:**

* Provide a user-friendly, intuitive interface
* Users must be able to navigate and use the application easily and efficiently.
* Display all appropriate information to the user
* Only information that should be displayed to the user is displayed as it is intended to be, when it is intended to be.
* Accept user input for processing
* The client-side must accept user input (e.g. button-clicks) and process the input accordingly.
* Perform regular checks on user input data
* Data from the user must be of correct type, size, and syntax.
* Send data to backend
* The client-side must be able to effectively and efficiently send information to the backend for processing.
* Perform regular checks on data being sent to the backend
* Data sent to the backend must be of correct type, size, and syntax.

**Constraints and Applicable Standards**

**Economic Constraints:**

* Budget
  + $100,000 budget for development of product
  + Limited to five developers for $87,000 salary each
  + Limited budget for purchasing licenses for new tools and technology

**Environmental Constraints:**

* Human
* The software should be directed toward both instructors and students. The design interface and operations should be such that a beginning software engineer can understand them easily.
* We have five group members and four months to finish this project. The scope of the project should reflect that. The quality will be excellent, but the complexity of the project will be limited.
* Equipment
* Because we are creating a standalone application, the equipment limitations will be set to the device the users are on while using the application.
* Technology
* We will be using several programming languages and tools in our application including but not limited to Node.js, JavaScript, Popcorn.js, CodeMirror, Bootstrap, HTML, and CSS. It is documented that each of these will work well with each other, so we should not have a problem ensuring the seamless integration of these languages and tools. More specific standards and references are listed in the “Third Party Library” section of this document.

**Availability and Support:**

* Our tool will be open source and released under MPL v2.0
* Firefox 33 and Chrome 38
* Browsers must have JavaScript enabled
* The application will be provided as-is and any technical support will be limited to communicating with the developers

**JavaScript Standards:**

The bulk of our coding implementation will be in JavaScript. This means we must adhere to strict standards of development to maintain legibility and understanding when examining portions written by other members of the team. Standard coding practices will be adhered to using sufficient commenting, proper indentation and naming conventions that make sense. That said, JavaScript has some quirks and different functionalities that are addressed below in the pursuit of good coding practice.

One of the biggest things to avoid, and one that is often ignored, is that the use of Global functions and variables should be kept to an absolute minimum. Using globals can cause major issues if you are using any external libraries. This is because external libraries may have functions or variables that have the same name and are also in a global scope. Because of the nature of JavaScript, multiple external files are common. Instead functions should be written in a namespace. This prevents scope issues but can cause the code to become a little wordier. We should also strive to avoid using JavaScript as a swap in for different technologies such as CSS. Doing so causes the code to be bulkier, and it makes it much more difficult for someone attempting to change certain aspects of the application. This is because they would have to go through the JavaScript to change the CSS.

One point of contention in JavaScript is when to use shortcut notation, as using or abusing the technology can cause readability issues for anyone who didn’t explicitly write the code. However, they can cut down on code and actually make it more legible if used properly (and in conjunction with proper comments). To this purpose we shall use shortcut notation primarily if assigning values to objects or arrays. This cuts down on the number of times the array’s or object’s name is written as well as making it more apparent what is being assigned when using objects.

**Third Party Library Standards:**

The third party libraries we will be using include:

* Node.js
* JQuery
* CodeMirror
* Karma
* Jasmine
* Bootstrap
* Popcorn.js

We will be using several external libraries for the development of our software. Node.js and CodeMirror are some environments/frameworks we will be integrating into our project. These environments and frameworks contain their own libraries that we will need to make use of. The JQuery library will provide some UI design and file upload capabilities. Karma and Jasmine will be the frameworks we use to test our software as we develop. Popcorn.js will be used to dynamically manipulate code segments. Twitter Bootstrap will also be used for the user interface of the web application.

There is extensive documentation online regarding the best practices and guidelines for using each of these frameworks and libraries. Rather than list each specific guideline, this section will simply layout some general information regarding the integration of these frameworks and libraries into our project and provide some URLs to useful references. It is important to note that all of the listed frameworks and libraries will work together. Here are some short, general guidelines for ensuring all of these frameworks and libraries play nicely together:

Jasmine and Karma will specifically be used for testing. All code should be written in a modular form in order to facilitate simple and efficient testing, with each test only testing a single component.

Bootstrap is specifically for the look and feel of the web application. It will be mostly used for HTML and CSS modification. Bootstrap has a specific library that may be included in the frontend design of the website where the developers feel it is appropriate.

JQuery is a popular JavaScript library and will therefore be incorporated right into the JavaScript.

Node.js will be used for the backend of our web application. It is based on Chrome’s JavaScript runtime and is intended for lightweight web applications.

CodeMirror is a JavaScript based text editor. This will be useful for making an uploaded file of code editable to student users should the developers to include it. Being JavaScript based, it should integrate easily into our program.

Popcorn.js will be used for code manipulation. It is a JavaScript library and, therefore, will also integrate easily into the rest of the JavaScript in our program.

Useful sources:

* <http://nodejs.org/api/modules.html>
* <http://api.jqueryui.com/>
* <http://codemirror.net/doc/manual.html#usage>
* <http://getbootstrap.com/getting-started/>
* <http://jasmine.github.io/2.1/introduction.html>
* <http://www.tuesdaydeveloper.com/2013/06/angularjs-testing-with-karma-and-jasmine/>
* <https://github.com/karma-runner/karma/tree/master/docs/intro>
* <http://popcornjs.org/documentation>

**Security**

**Cross Site Scripting:**

We are developing a standalone application. Any user input will be done offline and therefore should not be subject to XSS attacks. Also, our app will not be collecting any sensitive user data, which is usually the chief purpose of XSS attacks, and will primarily just be providing a service.

Normally, the chief way to mitigate this vulnerability is to sanitize any and all user input before it is used on either the client-side or the server-side. The user input our app processes is limited to duration of text-formatting actions, line numbers for some text-formatting actions, instructor comments, video/audio uploads, and code file uploads. We will perform checks on all user input to ensure it is the correct type of input. This should ensure that no malicious code will be uploaded.

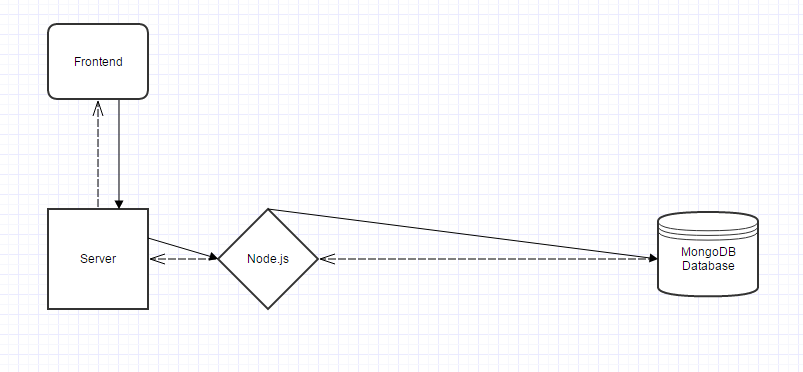
The largest threat that our design poses to the end user is after someone has constructed a page using malicious code with our tool and sends them the generated link. If the code that the constructor uploaded is malicious in nature, we could be handing over a readymade persistent XSS attack to the potential end users. We will escape all executable code, so that it is only uploaded as pure text. We will also notify end users to check that links are from a trusted source.

**Design**

**Backend – Node.js:**

The backend server will be built upon node.js using the default package and the npm service as required for extensibility. Its primary purpose will be to talk between the frontend and the server as well as serving content to the frontend application. The Node.js server must be able to generate a site from the frontend's input, store that site, generate a URL for that site, and return that URL to the frontend. It would also store the path and URL for the site in the database along with the owner of the site. This will allow the users to edit sites they have created as well as make it possible for the users to view a list of URLs for sites they’ve created. This will be especially useful if they lose one.

As illustrated below the frontend will make a request to the server on a Node.js port, which should usually be storing or retrieving content from the database, and the Node.js server will determine how to proceed. It can either return an error or a failure, or it could return some requested modification. It can also process the request and send it to the database. The database would then return the requested information, and the Node.js server would interpret the information. Node.js will then either serve the content back to the user or return a code that would indicate success or failure.



**File Uploads:**

In order to upload videos, audio clips, and code files, we have decided to use jQuery’s FileReader(). We will put the file upload functions in a separate JavaScript file as to keep it easily manageable and accessible. Video and audio file uploads will be fairly straight forward. We will just need to perform checks that they are the proper file types. The code uploads will be a little trickier, as CodeMirror loads new files without getting rid of old ones by default. To get around this, we will need one function that acts as soon as the user clicks the code upload button. This function will immediately remove the current CodeMirror instance regardless of whether a new one is loaded. Then if the user does select a new code file to load, a new instance of CodeMirror will be created. If the user instead chooses to cancel the file upload, the original code file will be reloaded back into the CodeMirror instance.

**CodeMirror:**

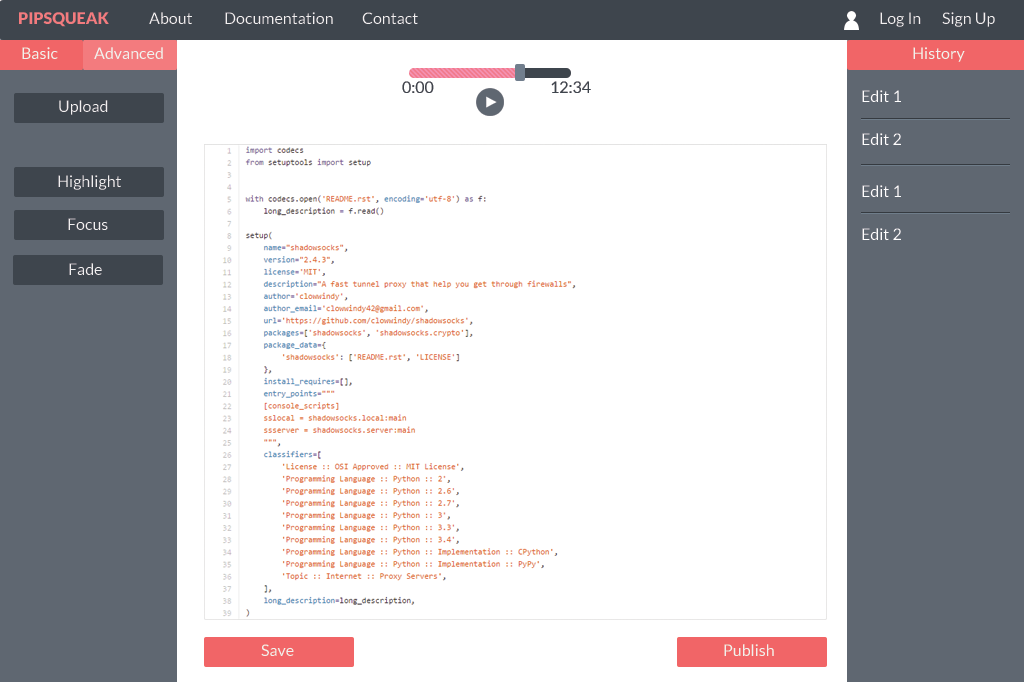
To more effectively teach programming, the instructor will want to work in an editor with features that provide a visual aid to the student looking at the code. For this purpose, CodeMirror will be used as the text editor in the programming lecture tool. CodeMirror is a specialized text editor that is implemented in JavaScript for the browser. It is an open source project shared under an MIT license. An advantage to using CodeMirror is that it supports many different languages (over 60), which gives the lecture tool versatility. There are many useful features that will be valuable for this software. Syntax highlighting will allow the instructor to highlight sections of the code when discussing them. Code folding will allow for users to hide unimportant or already discussed portions of the code so that one can focus more on the current lines being taught. Bracket and tag matching will make it easier for students to see the beginning and end of loops, functions, etc. Line-number gutters organize the code so that instructors can refer to lines by number in their lessons. CodeMirror also supports various themes, font styles, and font sizes to allow for more visually appealing and creative lessons.

**Popcorn.js:**

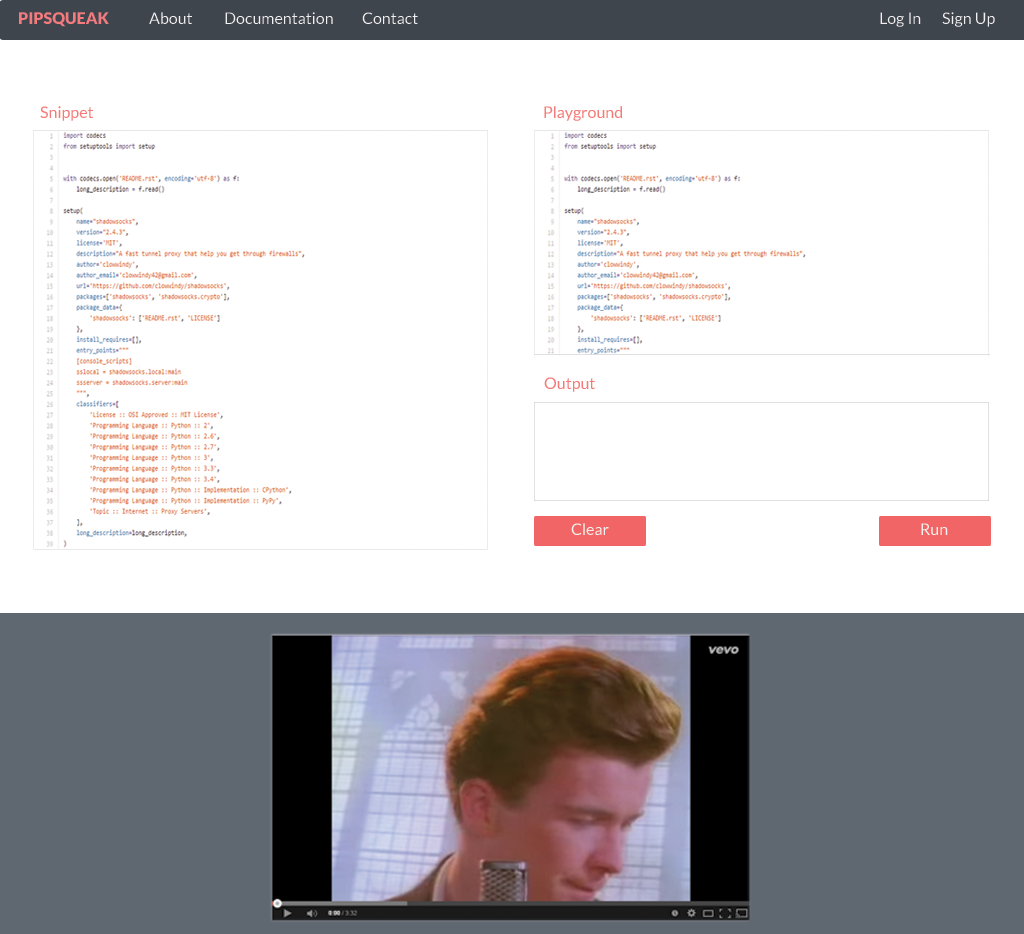
With the programming lecture tool, instructors will have the ability to create recorded video lessons while interacting with their code to more effectively teach students. In order to make these video recordings possible, we will use the Popcorn.js API. Popcorn.js is an open source JavasScript library created by Mozilla. It is available for free under the MIT license. Using Popcorn, the interactions with the code can be mapped to a timeline. Instructors will upload an audio file, and Popcorn will match up the events and the audio chosen by the instructor to the desired time in the media. By including the Popcorn.js library, the most important functionality of the lecture tool is unlocked. Using the methods in the API, we can easily manipulate the webpage based on the current position of the video. This will allow for instructors to create a rich, interactive learning experience for students.

**UI Mocks:**

Professor View



Student View

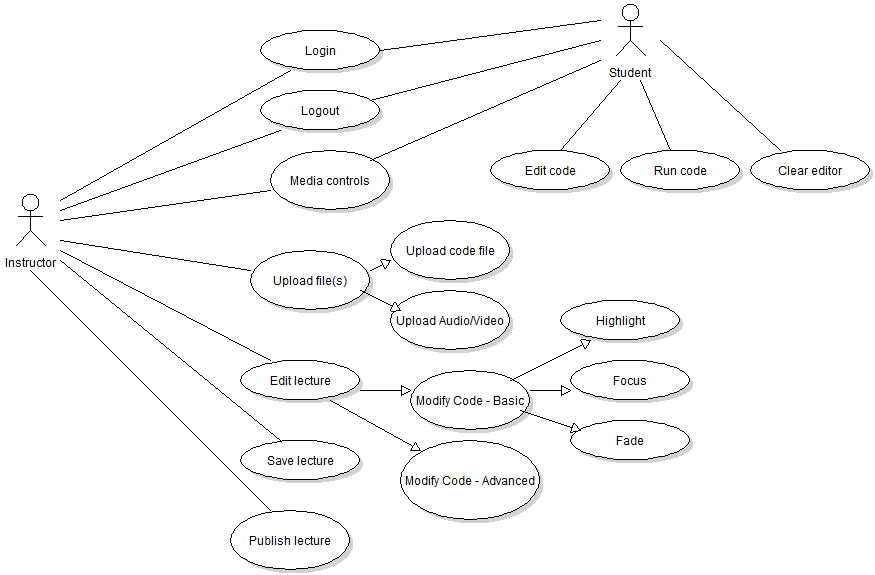


**UML Activity Diagrams:**





**Use Case:**

****

**Testing/Data Collection and Analysis Plan**

**Testing Plan**

As a team we decided to use a mild form of Test Driven Development (TDD) that will essentially require us to write unit tests for each block of code we write. The idea here is that the Scrum Master and Product Owner will work together to create acceptance criteria, and any code written for a task must have an associated test that meets the given acceptance criteria. This alteration to TDD allows us to focus on writing the code that is needed while still ensuring that testing is done in all aspects relevant to the overall program. As the code base grows, so too will our tests, and we can rest assured that changes in the future won’t break code from the past (and if it does we can address it immediately and be aware of the conflict).

To achieve this goal we will not accept any task as complete unless the code works and all acceptance criteria has been validated with a unit or integration test. We plan to use JavaScript based testing frameworks to facilitate the testing functionality of this requirement. We will require that all team members be familiar with the framework in order to better understand what any given test is accomplishing and to provide more eyes on the code being written. This leads into the next testing component, and that is the proofreading of anything developed. In addition to the tests written, another team member will be required to review the code and the test before moving the task into the “Testing” column of our work board. Anything placed in the work board will be vetted by the developer and the proofreader, and then the Scrum Master or Product Owner will review it once more. This will be done before either accepting it or adding notes on why it can’t be accepted yet (which will return to the developer to fix and repeat this process).

For frameworks, we are planning to use a combination of Karma and Jasmine testing frameworks, as they both work well together to perform unit and integration testing on JavaScript based programs. The additional perk that they bring is they also integrate nicely with both of our chosen development frameworks: AngularJS and Node.js. Additionally, since we chose to write our entire code base in JavaScript, we should be able to have maximum code coverage using just these two testing frameworks.

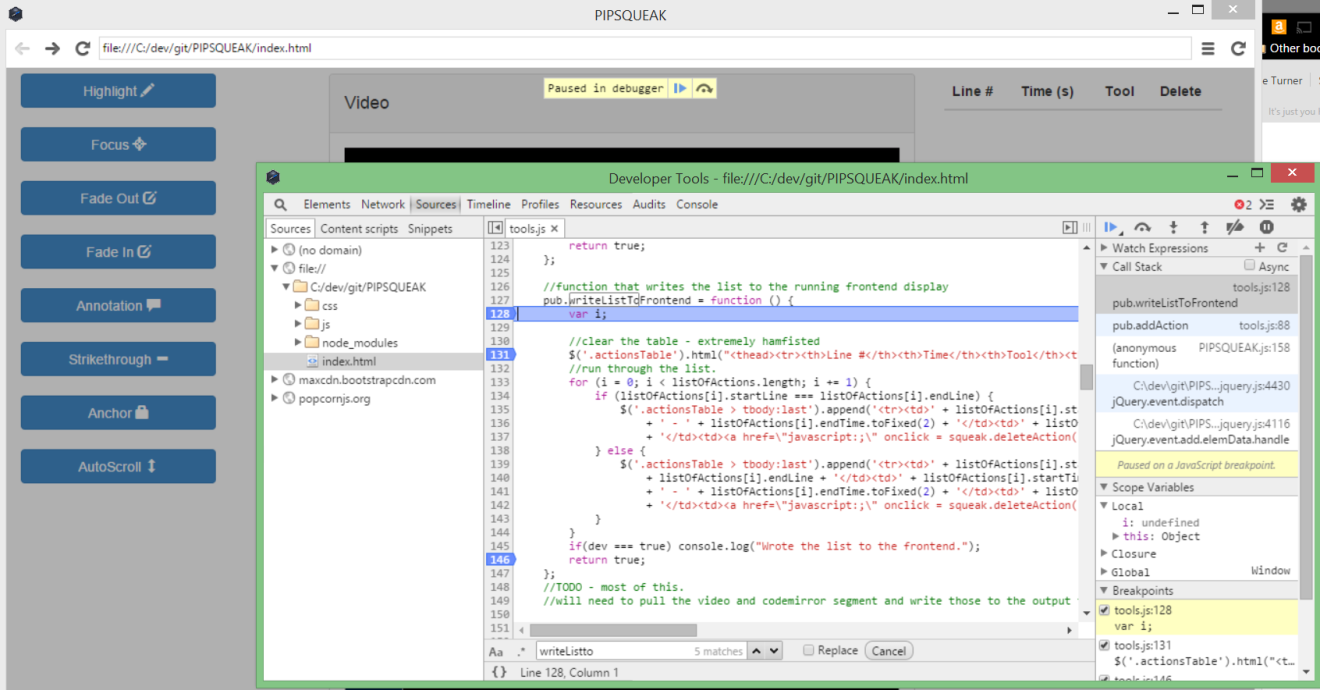
One drawback to the Jasmine and Karma testing frameworks is their incompatibility with JQuery. JQuery will be required in certain areas of our project. The Jasmine and Karma testing frameworks don’t always work well with JQuery, so for the parts of our project that use JQuery we will need to use an alternate form of testing. For methods that call another method using JQuery we will simply mock the method that uses JQuery so we can continue to validate the current method we are testing. For the method that uses JQuery, however, we have, as a group decided that the best way to test these parts of our code is by assigning someone to manually step through the execution of the function in that portion of code in our program. This touches on the proofreading concept mentioned earlier. Essentially, by assigning a specific team member to perform each specific function in our project, we hope to ensure that the intended result of each function is properly carried out without error.

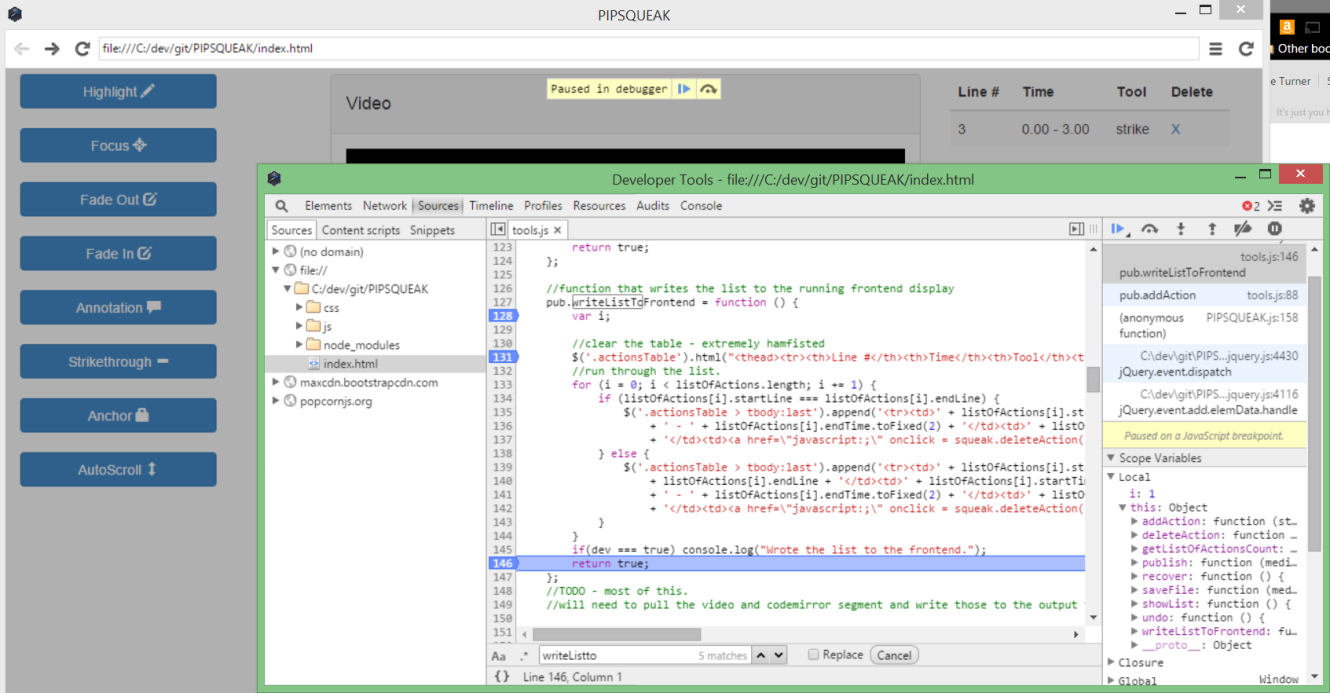
Finally, the types of things that need to be tested should be declared. Each portion of code that needs to be written for our project will be specified in an associated task through the JIRA project tracking software we are using for project implementation. One team member will be assigned to each task. Then upon that task’s completion, another team member will review the task to ensure it is complete. This process guarantees testing is performed on every bit of code put into our project. Then some of the more important functions, if they are compatible, will be tested using the Jasmine and Karma testing frameworks mentioned earlier. Important functions include any function that is critical to the successful use of our software. If these important functions do not operate as intended, the software will ultimately fail. If these functions are not compatible with Jasmine and Karma, like functions using JQuery, then they will be tested using the peer review method described earlier.

Below is the output of the current Unit tests we have written:



An example manual test that we perform is for the method "writeListToFrontend()." This method uses JQuery and as such requires manual testing. For these tests we load the application and manually step through the function using breakpoints as shown in the below screenshots. As we step through the method we check the variables along the way and ensure that each piece of the method gets called appropriately and that the end result is returned correctly.





For each test we document the results in a similar manner to how we write our Jasmine/Karma tests. For writeListToFrontend() we write:

"writeListToFrontend()"

"should append to the action table for a single line" : passed

"should append to the action table for multiple lines" : passed

"should return true" : passed

**System Performance, Testing and Evaluation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | **Expectation** | **Actual Result** | **Corrections** | **Comments** |
| **Hardware** | **Compatibility** | 100% success on modern x86 x64 systems | Success | none | Mac, Linux, and Windows based hardware worked |
|  | **Performance** | Audio/Video plays without stuttering | Success | Machine Dependent | retested and final test returned 1.8 sec response time |
|  | **Reliability** | zero down time | Success | none | all hardware operated successfully with no down time during the test |
|  | **Backwards Compatibility** | Works on pre-modern hardware | Success | none | Tested on rasberry pi and Windows XP machines |
|  |  |  |  |  |  |
| **Software** | **Compatibility** | no compatibility issues with operating systems | Success | none | Tested OSX, Windows 7/8, Linux Ubuntu |
|  | **Performance** | Upper limit of publish time not to exceed: 1 minute | Success | none | Publication with up to 1gb A/V file will build the publication within 1 minute. |
|  | **Maximum File Size** | File size performs well up to 1GB | Success | none | System can handle video up to 1GB. Maximum publish time before this system runs out of memory is 5 seconds |
|  | **File Formats** | Works with common internet formats Except: MP3/MP4 without extensions | Success | none | Supports ogg, mkv, and wav file formats natively. Needs GPL based H.264 extension for MP3 and MP4 |
|  | **File Upload** | File upload gracefully handles improper file types | Success | none | File upload will load the files regardless of type and will not crash the system. |
|  | **Code Length** | Code file upload supports code up to 1000 lines | Success | none | Code Editor supports beyond 1000 lines |
|  | **Number of instructions** | Can support up to 3,000 instructions | Success (See Comment) | none | Each line for an instruction call counts as an instruction. |
|  | **Published View** | Works in Chrome browser | Success (Firefox works also) | none | Works in Chrome browser version 41 and above as well as Firefox version 35 and above |
|  |  |  |  |  |  |
| **Implementation** | **User** | Novice user can utilize the system with 1 hour of training | Success | none | Novice user can use the system with minimal training |
|  | **Navigation** | Clicking Publish automatically loads into the published view | Success | none | Works as expected |
|  | **Error Handling** | Display Error message without the program crashing | Success with modification | In NW.js many errors would automatically trigger a page refresh. We had to redirect the refresh event to an error pop up instead. | Error messages are displayed and the application does not unexpectedly crash |
|  | **Testing Framework** | Jasmine/Karma Unit and Integration Tests | Partially Successful | Karma tests would not effectively test our code since we don't make any web calls. As a result we had to manually test a lot of functionality. | Created unit tests using Jasmine on top of Karma, but we opted to manual testing for integration tests |
|  | **Standalone** | Does not require any outside internet calls | Successful | none | Works in a self contained environment without requiring any web service calls |
|  | **Outputs** | Published Folder/Zip | Partially Successful | Compressed the output into its own directory structure for portability, but with no zip functionality (nw.js issues). | Split out the save and publish features to work similarly, but with save creating a Zip. |
|  | **Inputs** | Video/Audio/Code files are taken as inputs | Successful (see Comment) | Restricted the Audio/Video format types to ogg, mkv, and wav | MP3 and MP4 require a GPL based plugin |

**Testing Plan for the Project Report:**

Writing the project report was conducted by splitting all of the necessary parts of the paper into specific tasks and distributing them over weekly milestones throughout the semester. The tasks and the milestones are documented in the “Timeline” section of this report along with the due dates for each milestone. Each task is then assigned to a group member so that each group member contributed equally to the final report. Testing the project report was then done by assigning a proofreader to each group member. The proofreader was another group member responsible for ensuring proper grammar, spelling, and content within each task. If something needed correcting, the proofreader added a comment to the task. The creator of the task then added the necessary changes and resubmitted the task. The task was then proofread again, and if all was well, the task was considered complete.

**Project Management Plan**

**Methodology:**

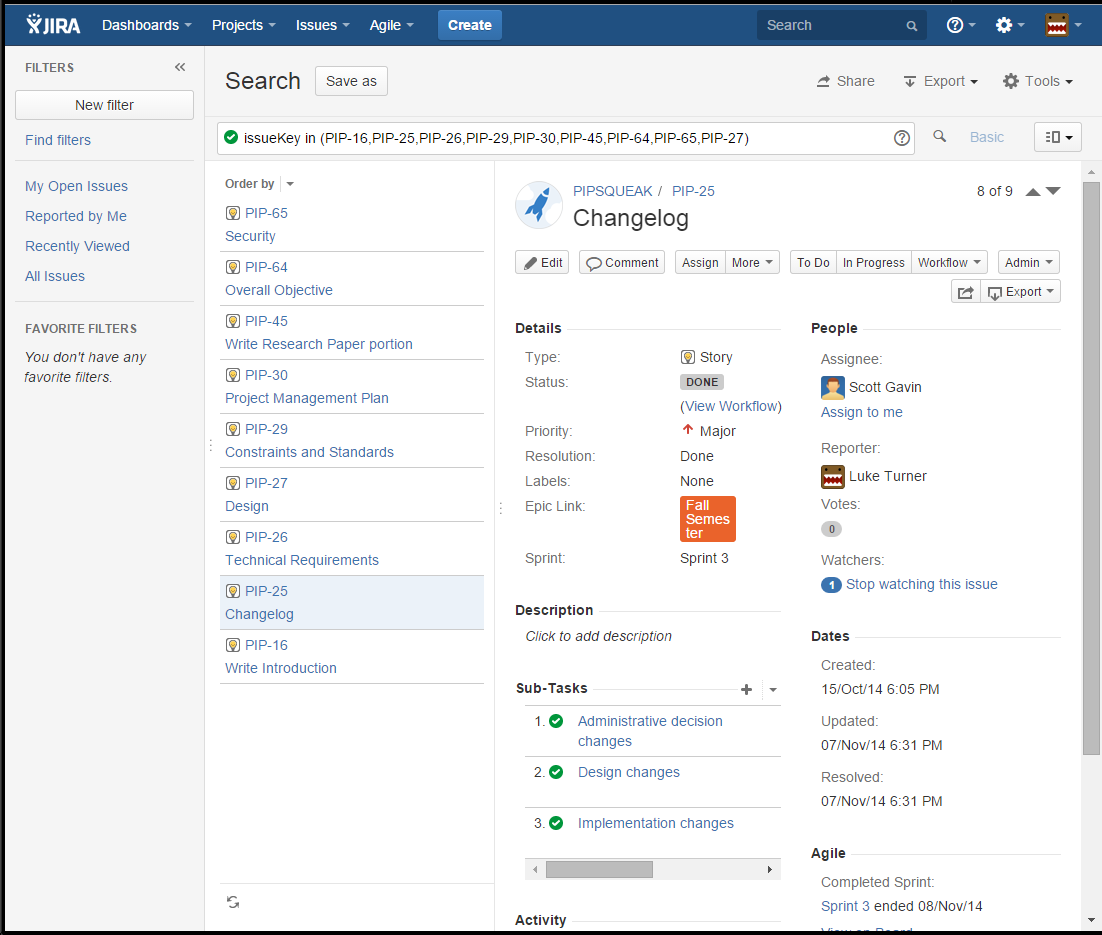
Team PIPSQUEAK decided to use the Agile Scrum methodology for organizing our team’s efforts toward both the project design document and the software development next semester. For our Scrum team we decided to have Luke be the Scrum Master. Andrew is taking the role of Product Owner. Avery, Scott, and Bill will take the role of Developers. Most of us are working and taking large course loads both semesters, so we plan to have our stand up meetings at least twice a week with one week long sprints (we will adjust this to two week long sprints once we hit software development). Sprints will start on Thursday and end the following Wednesday with a stand up, followed by a retrospective, and then finally the planning meeting for the next sprint.

For our stand ups we have decided to essentially discuss what has been done and what is to be done each time that we meet. We may, down the road, implement a web based stand up for the days that we don’t meet, but this will only be enabled if a sprint goes badly and will be used as a Scrum Master tool for destroying impediments. For retrospectives we will try to iron out and estimate errors (too high or too low) from the previous sprint and discuss the good and the bad for the stories tackled that sprint. The Scrum Master will take note of any impediments that arose from the retrospective and resolve them in a prioritized manner during the next sprint. As for the planning meeting, we will assess the list of stories prepared by the Scrum Master and the Product Owner and score them by agreeing on a number based (Fibonacci) weighting system. Using this system, we can establish our team’s velocity and appropriately estimate and plan future sprints for our team.

For management tools our team is using Atlassian JIRA, Confluence, and Stash file storage, Scrum story planning board, and document continuation and tracking. We are also using Google Plus and Hangouts to communicate within our group. Some possible tools we may also use in the future include GitHub, Microsoft Onedrive (Word), and possibly Babbage if it will allow the frameworks we are using and be flexible enough for our design.

The reason our team chose Agile’s Scrum methodology is because all of us have used it previously in Software Engineering. Also, a few of us currently use it at our workplaces or have used it in an intern environment. This familiarity saves us a lot of time by not needing to spin up on a new method or toolset to accomplish the tasks laid out before us. Lastly, of the many options out there for organizing a loose group of five into consistently delivering and constantly upkeeping, Agile Scrum allows the most flexibility and provides some valuable risk management in the form of sprint deadlines, Scrum Master, Product Owner, and retrospective meetings.

An example work board from JIRA includes our Scrum stories and tasks as seen in this Sprint 3 report:



**Project Management Plan Timeline:**

***Fall Semester 2014***

* **Milestone 1 Due 10/08/2014**
  + Determine collaborative development tools to be used
  + Find mentor
  + Decide on group name
  + Determine development method
  + Decide what problem our software will solve and the general idea regarding how the software will solve that problem
  + Set up JIRA
* **Milestone 2 Due 10/15/2014**
  + Decide on research topics
  + Edit bios to be included in project report
  + Cover page
  + Create testing plan
  + Create title page
* **Milestone 3 Due 10/22/2014**
  + Research papers
  + Cost Analysis
    - Personnel Costs
    - Non-Personnel Costs
    - Total Expected Costs
    - Running Costs and Upkeep
    - Graphic
  + Project Management Plan
    - Scrum Methodology
    - Timeline
    - Required Independent Learning
  + Testing/Data Collection and Analysis Plan
    - Unit testing framework
* Problem Statement
* **Milestone 4 Due 10/29/2014**
* Change log:
* Administrative decision changes
* Design changes
* Implementation changes
* Security:
* XSS
* Write Introduction
* Executive summary
* Problem statement
* Research problems
* Reference Page
* Glossary
* Design
* Discuss design
* Backend: Node.js
* Popcorn.js
* **Milestone 5 Due 11/05/2014**
* Technical Requirements
* Server requirements
* Client-side requirements
* Constraints and Standards
* Economic Constraints
* Environmental Constraints
* Availability and Support Constraints
* Standards JS and other languages
* Standards for database
* Standards for third party libraries
* Overall Objective
* **Milestone 6 Due 11/12/2014**
* Documentation
* Add existing components
* API
* FAQ
* Licenses and adding for Git commits
* Progress reports
* Development progress report
* Development Report on Paper/Design (sprint by sprint)
* Testing Plan
* Project Report/Design
* Development
* Other
* Table of Contents
* Graphics mocking out design
* Results Section
* Discussion Section
* Finish rough draft of report
* **Milestone 7 Due 11/19/2014**
* Make any necessary edits and additions
* Add graphics to report
* Finalize project report
* Make paper flow with one voice

***Spring Semester 2015***

***Frontend – Bill and Avery***

***Backend – Luke, Scott, and Andrew***

* **Milestone 1 Due 02/11/2015**
  + General – Regroup and reestablish group communication and dynamics
  + Frontend – basic instructor view layout
  + Backend – basic file system setup
  + Backend – merge frontend into NWJS
  + Testing of all milestone acceptance criteria
* **Milestone 2 Due 03/18/2015**
  + Frontend – add audio/video/code file upload button
  + Frontend – add playback capability
  + Backend – add audio/video/code file upload capability
* Frontend – create highlight button
* Backend – add highlight capability
* Frontend – create button to focus on certain lines (removing other lines temporarily)
* Backend – add focus capability
* Frontend – create annotation popup button
* Backend – add annotation capability
* Frontend – create button for global control strikethrough
* Backend – add strikethrough capability
* Frontend – create button for global control fadeout
* Backend – add fadeout capability
* Frontend – create button for global control fade-in
* Backend – add fade-in capability
* Frontend – create button for global control anchor
* Backend – add anchor capability
* Frontend – create button for global control auto-scroll
* Backend – add auto-scroll capability
* Backend – store events
* Testing of all milestone acceptance criteria
* **Milestone 3 Due 04/22/2015**
* Frontend – create student view
* Backend – playback events
* Backend – file generation upon successful completion of tutorial
* Frontend – student view tutorial video controls
* Testing of all milestone acceptance criteria
* **Milestone 4 Due 05/05/2015**
* Frontend – Presentation
* Backend – Presentation
* Frontend – Only minor alterations if absolutely necessary
* Backend – Only minor alterations if absolutely necessary

**Cost Analysis**

**Industry Costs for Mid-Missouri:**

Based on reports from sites such as SimplyHired.com and Salary.com, the average salary of a software engineer in Mid-Missouri is about $87,000. For a team of five developers, the cost of labor would be about $435,000. Based on current available rates, office space in Columbia can be rented for around $15 per square foot.

**Estimated Development Costs:**

Our team will require several resources during the production and maintenance of our lecture creation tool. For the most part our required resources are considered fairly standard for a software development team. The majority of our costs will come from office space and upkeep. In addition to the costs incurred from our team’s required resources, we will also need salaries and reasonable benefits comparable to what a software engineer working in the industry would receive. These costs also take into account the cost-of-living in Columbia, MO.

**Required Resources:**

* *Server Space: $171.00/month*We need server space to host JIRA for our agile/scrum project and issue tracking. A server will not be required for our application.
* *Atlassian: $30.00/month*To effectively execute our agile development methodology, we will use two Atlassian products: JIRA and Confluence.
* *Office Space: $1,875.00/month*The team will require office space that can comfortably accommodate five people and their workstations so that we may collaborate together.
* *Workstations: $9,000.00*Workstations suitable for writing and running code will be required. This cost includes the OS and all necessary development software that we will be using.
* *Utilities: $1,000.00/month*In order to keep our office up and running, we will need light, water, electricity, air conditioning, and heat.
* *Internet: $80.00/month*It is imperative that our team has access to a fast internet connection that can support five simultaneous connections with medium - high usage.
* *Legal: $1,000.00/month*We will employ a legal team that we may consult with any legal concerns we may have. The team will also be able to fight for us in any potential infringement cases.

Upfront cost: $9,000.00

Total monthly cost: $3,996.00  
Total yearly cost: $47,960.00

**Employee Salaries and Benefits:**

* *Salary: $87,000.00/year*Each member of our team will be paid based on the average salary for a software engineer/web developer working in Mid-Missouri.
* *Benefits: $26,100.00/year*On top of salaries we will also account for benefits for each employee. Benefits accounted for here include health insurance and retirement.

Monthly Cost: $47,083.33

Yearly Cost: $565,500.00

**Required Independent Learning**

*\*It should be noted here that we decided against using AngularJs during development, but we did carry out this required learning task before making that decision.*

The team will be learning AngularJS via a free course at:

<http://campus.codeschool.com/courses/shaping-up-with-angular-js/intro>. The site offers 5 courses that cover the majority of the AngularJS functionality, so we should all come back from Winter Break ready to code using AngularJS to its full potential.

**Results**

Main application view

Entering duration of action on selected lines



Main published view

Published view with actions running on code

For our final implementation we built an NW.js (Node Webkit) application that allows instructors to annotate and manipulate code that accompanies a video lecture. The manipulation actions are executed based on the video timestamp. For example, the instructor could select a section of code when the video is at 0:05 seconds and set the code to highlight for a duration of 5 seconds. In this example, when the video is played in the published view and reaches the 0:05 second mark, the selected piece of code will be highlighted for 5 seconds, until the 0:10 timestamp. Pipsqueak can also publish created lectures to folders containing an HTML file that can be viewed in most web browsers.

As a result of the tool being built upon the NW.js framework, our tool can be used on both Windows and Mac machines as long as the user has Node.js installed. The screenshots above showcase the final page designs of the creation/editing view as well as the published view.

Pipsqueak has support for seven different actions that can be applied to selections of uploaded code for a set duration based upon the current video timestamp. These seven functions are: highlight, focus, fade, annotate, strikethrough, collapse and scroll. The highlight function highlights the selected code with a yellow background, identical to what you may do if you were highlighting code found in a textbook. Focusing causes the code to increase in size. Fading the code will cause it to disappear for the duration of the function. The annotate function adds an indicator to the code that contains a user supplied link that could help further explain that section of code. Strikethrough draws a line through the code but still displays it. Collapse removes the selected code so that two sections of code that are not adjacent can be discussed at the same time. Finally, the scroll function will auto-scroll the code to the lines that are currently being manipulated by an action. This allows the student to sit back and watch instead of having to struggle to scroll to the next bit of code that the professor is talking about.

Because our application is built using NW.js, we were able to write code similar to what we would have written for a normal web application. This means the majority of our code was written in JavaScript with some additional CSS and HTML. In addition to the code we authored, we used several third party libraries including: Twitter Bootstrap, CodeMirror, jQuery and jQuery UI, Google Code Prettify, and Popcorn.js.

**Discussion**

This section will be written after the project is completed in Capstone 2. It will contain our discussions about why and how we did what we did, whereas the results are more what we actually did. The discussion section is primarily intended to give rationalizations for design decisions and implementation decisions.

**Future Work**

The future work section will include any plans for continuing the project or extending the project beyond the scope of this document and the work completed during Capstone 2.

**Appendices**

**Frontend Frameworks Research – William Moors:**

We want to provide the best possible experience for the user. In order to do that, we will need to decide on a frontend framework that allows us to include all of the required functions of our project. Additionally, this must be done in a way that will be aesthetically pleasing and intuitive to a beginning user. There are a plethora of frontend frameworks available online. It will take some careful consideration to choose the best fit for our project. Some criteria the framework must fit are: 1) open source, 2) good online support community, 3) JavaScript compatible, 4) audio/video support and 5) works with Popcorn.js. With that in mind, there are a few frontend frameworks (some just for testing) that we would like to include in our research. These frameworks include AngularJS, Ember.js, Jasmine, Karma, and Bootstrap.

***AngularJS***

AngularJS was developed in 2009, and it has grown exponentially over the past few years. Now backed by Google, AngularJS has an incredible amount of online support. There are tutorials, FAQ, question/answer forums, and tons of online discussions on sites like [www.stackoverflow.com](http://www.stackoverflow.com). This online presence will help us immensely should we choose AngularJS as a frontend framework for our project. Another huge positive for AngularJS is that it is compatible with many other frameworks. Although there have been reports of difficulty, AngularJS can play nicely with Twitter Bootstrap. This is an exciting possibility because we would be able to use AngularJS’s JavaScript savvy DOM manipulation along with Bootstrap’s sleek CSS. AngularJS also supports audio/video media playback. This can be done through a variety of libraries, but it seems like FlowplayerJS and HTML5 media functionality are two of the most popular ways of integrating media playback with AngularJS. On top of that, Karma and Jasmine can be used as testing frameworks to test AngularJS code. This could potentially be very useful for our project when we enter the coding phase. Writing Karma and Jasmine tests for our AngularJS framework would help tremendously by allowing us to dynamically troubleshoot our code, saving us time in the long run.

AngularJS has some shortcomings as well. AngularJS has a feature called two-way data binding. This essentially saves backend developers a lot of code-writing by allowing frontend developers to simply render templates straight into the HTML. This is a good thing, but how it is accomplished may be very expensive and can substantially slow down a web application. AngularJS does its two-way data binding by detecting changes in the model and automatically updating the view through integrated logic in the controller. This process has been dubbed “dirty-checking” and is different than the “trigger listeners” that Ember.js employs. It can be expensive because it compares updated values with previous values every time a change occurs. Another shortcoming of AngularJS that I can see is that it does not follow standard MVC architecture. While it uses a model, view, and controller, it also uses directives, factories, filters, and services. These elements are intended to make AngularJS easier to use, but there may be a bit of a learning curve to those of us that have not used it before.

***Ember.js***

Ember.js was originally developed in 2007 by a company called SproutIt, but was not named Ember.js until 2011. Ember.js also has a lot of online support. Tutorials and community support forums can be found on websites like [www.emberjs.com](http://www.emberjs.com), [www.stackoverflow.com](http://www.stackoverflow.com), and [www.youtube.com](http://www.youtube.com) among others. It seems that AngularJS has a bit more online support, but Ember.js has a large online presence as well. Like AngularJS, Ember.js is compatible with a variety of other potentially helpful frameworks. Ember.js works with Bootstrap to create professional, user-friendly user interfaces. Numerous tutorials and documentation supporting this relationship can be found online. One website is solely devoted to the relationship between Bootstrap and Ember.js. That website is <http://ember-addons.github.io/bootstrap-for-ember/>. Ember.js also plays nicely with HTML5 and a large file uploader called Plupload. These are two potential ways we could incorporate uploading audio and video files on our website. As the case was with AngularJS, the Karma and Jasmine frameworks may also be used with Ember.js in order to logically and incrementally test our application.

The difference between Ember.js and AngularJS lies primarily in two areas: 1) two-way data binding implementation and 2) MV\* architecture and intuitive design. Both of these two areas were listed as negatives for AngularJS, but I am going to list them as positives for Ember.js. While AngularJS implemented its two-way data binding through a process called “dirty-checking”, while Ember.js does this through a process called “trigger listeners”. Trigger listeners are basically a way of updating the DOM based on events firing or misfiring. While this can potentially pile up on your stack, this problem can be avoided with proper coding – namely ensuring that the event changes triggering the listeners do not continue to change more events and trigger more listeners. AngularJS had a slightly more complicated architecture including model, view, controller, directives, factories, filters, and services. Ember.js prescribes a little more to the standard MVC architecture, making it easier for beginners to grasp.

***Jasmine***

Jasmine is a framework for testing JavaScript. This could come in handy because we are planning a very JavaScript-heavy application. It will work with both Ember.js and AngularJS and should serve us well as we logically and incrementally test our code throughout development. Jasmine was developed to be very easy to read and is heavily documented on the official website, <http://jasmine.github.io/>.

***Karma***

Karma is a JavaScript testing tool. It works with testing frameworks like Jasmine to create executable JavaScript code in “multiple, *real* browsers.” It supports all major browsers and will work well with both AngularJS and Ember.js. Karma has online support at <https://github.com/karma-runner/karma> and <http://karma-runner.github.io/0.12/index.html>. Karma and Jasmine should give us a simple way to test our JavaScript code as we go.

***Twitter Bootstrap***

Twitter Bootstrap is a frontend framework like AngularJS and Ember.js; however Bootstrap is geared more for design and user interface and less toward JavaScript and MV\* architecture. While AngularJS and Ember.js are intended to dynamically link frontend and backend development to make things easier on developers, Bootstrap is more about giving developers an easier way to make their websites more aesthetically pleasing and easy to use. Bootstrap can be used along with both AngularJS and Ember.js. Although this may not be necessary, this would provide an easier way for us to design our website. It is also important to note that, like the other frameworks mentioned, Bootstrap is heavily documented online. There are tutorials and support forums on several websites including <http://getbootstrap.com/2.3.2/> and [www.stackoverflow.com](http://www.stackoverflow.com).

***Overall***

Karma and Jasmine would be good to use together in order to provide an easy way to test our code throughout the development process. Bootstrap is a good option for creating a sleek, user-friendly interface. Ember.js and AngularJS both provide good ways to implement a JavaScript-heavy web application. They both have extensive online support, but AngularJS has a little more than Ember.js. They both provide two-way data binding for dynamic Model-View alterations. AngularJS does this through dirty-checking, comparing old values to new ones. Ember.js implements two-way data binding through trigger-listening, making updates after event fires and misfires. Ember.js seems to have a slight upper hand in this difference, as comparing old values to new ones every time they are updated can cause a much slower, more aggravating experience for the user. Ember.js has a slight advantage over AngularJS in terms of a more intuitive design for developers. Ember.js has a more standard MVC architecture, while AngularJS incorporates a few more concepts that cause a higher learning curve for new developers.

**References:**

1) <http://en.wikipedia.org/wiki/AngularJS>

2) <http://en.wikipedia.org/wiki/Ember.js>

3) <http://www.airpair.com/js/javascript-framework-comparison>

4) <http://andyshora.com/unit-testing-best-practices-angularjs.html>

5) <http://angulardirectives.joshkurz.net/dist/#/flowplayer>

6) [http://en.wikipedia.org/wiki/Jasmine\_%28JavaScript\_framework%29](http://en.wikipedia.org/wiki/Jasmine_%2528JavaScript_framework%2529)

7) <https://github.com/karma-runner/karma>

**File Uploading Research – Andrew Dominick:**

The programming lecture tool is designed as an aid to teaching students and novice programmers how to code. To teach code effectively, one would obviously prefer to see some actual code. Instructors using the tool must be able to provide the code necessary to teach the desired lesson. For this reason, file uploading will need to be a functionality of the software. There are several different approaches to file uploading worth exploring to find the best fit. Important factors to consider include security, ease of usability, and best practices. External libraries should also be considered for implementing file-uploading functionality.

One of the most important aspects of software is security, and file upload forms can pose a major security threat. Allowing end-users to upload files to a web application opens up the door for pernicious users to compromise the server. Vulnerabilities from file upload forms are quite common, as the people at Acunetix discovered while testing several different web sites. Their testing showed that many applications use a simple HTML form with a PHP script. A common vulnerability is not restricting what types of files users are allowed to upload. Attackers could upload a PHP or .NET script with malicious code that could cause havoc to the application. Another error found was improperly using mime type validation. Attackers could upload scripts that allow for the sending of HTTP POST requests, allowing them to send a fake mime type. Acutenix found that a popular strategy for securing file upload forms is to protect the file where files are uploaded within the .htaccess file. This is not a secure approach because attackers can overwrite the .htacess file with their own modified version. Client-side validation is also common in file uploads, which can be bypassed by malicious users through their own client-side scripts that do the validation.

To prevent the security issues and provide the best user experience possible, the file upload form should follow the best practices and guidelines laid out by other programmers who have experience in the subject. On the Sans Software Security blog, Johannes Ullrich described some general best practices for file uploads that the software engineering community has realized. It is suggested that a unique, unpredictable file name like a hash is used on your local system, as opposed to using the name supplied by the user. The uploaded file should be stored in a directory outside of the document root so that attackers cannot access the file directly. Another key procedure is to set a maximum file size for the upload form. The code should check the file size for each upload to make sure it doesn’t exceed the maximum size. The amount of files allowed to upload to the tool should also be limited.

The interface to upload a file should be easy for users to understand and look visually appealing. For this purpose, the DropzoneJS library is a viable option. As it is described on its website, “DropzoneJS is an open source library that provides drag’n’drop file uploads with image previews.” Implementing DropzoneJS, users are able to drag and drop the desired file into an area on the page called the drop zone. The file will then be uploaded with an image preview and ready to store. This is especially useful for this software if the user wants to upload multiple code files at once. Some other benefits of the library are that it is lightweight, it does not depend on any other library, and it can be highly customized. The features of DropzoneJS and its detailed documentation make it a good option for implementing file uploading.

Another open source file uploading library available is the jQuery File Upload widget from web developer Sebastian Tschan. It is a jQuery plugin that supports file uploads with any server-side platform. The features described on the plugin’s download page include multiple-file selection, drag and drop support, progress bar, validation and preview images, and audio/video. The interface consists of four buttons - add files, start upload, cancel upload, and delete. There is also a jQuery UI version available for use with the same features. The disadvantage of this library compared to DropzoneJS is that it has a dependency on the jQuery library. An advantage is that the interface is more simple and practical than that of DropzoneJS. The jQuery File Upload widget would be very useful for implementing file upload functionality on the programming lecture tool web application.

The programming lecture tool will require file-uploading functionality so that users can upload code to teach others from. In researching the topic, a number of security concerns were discovered. In development, those security issues can be avoided by implementing the numerous precautions stated in this section including file authentication and limiting the file size/type. The file upload form is desired to be easily understandable and practical for end users. External libraries were researched and the DropzoneJS library and the jQuery File Upload widget were found to be useful tools for development. The jQuery File Upload widget appears to be the best option for its ease of usability and ease of implementation.

**Sources**

* Calin, Bogdan. "How File Upload Forms Are Used by Online Attackers." *Acunetix*. N.p., May 2009. Web. 27 Oct. 2014. <http://www.acunetix.com/websitesecurity/upload-forms-threat/>
* Meno, Matias. "Dropzonejs." *Dropzone.js*. MIT, 2012. Web. 27 Oct. 2014. <<http://www.dropzonejs.com/>>
* Tschan, Sebastian. "JQuery File Upload." *JQuery Plugin Registry*. MIT, n.d. Web. 27 Oct. 2014. <http://plugins.jquery.com/blueimp-file-upload/>
* Ullrich, Johannes. "8 Basic Rules to Implement Secure File Uploads." *Software Security with Frank Kim*. SANS, 28 Dec. 2009. Web. 27 Oct. 2014. <http%3A%2F%2Fsoftware-security.sans.org%2Fblog%2F2009%2F12%2F28%2F8-basic-rules-to-implement-secure-file-uploads>

**Core Libraries: Popcorn.js and Highlight.js Research– Avery Wells:**

Our lecture creation and student learning tool will extensively utilize two different third party libraries: Popcorn.js and Highlight.js. These libraries will help to present code segments in a way that makes them easier for students to understand. Here, the advantages of using each library will discussed as well as how to use them in regards to our project.

***Popcorn.js***

Popcorn.js is an open source JavaScript library for creating time-based interactive media on the web and is available under the MIT license. [1, 2] Popcorn.js is able to create media elements by taking the preexisting HTMLMediaElement interface, which has properties and methods common to all media-related objects, and turning it into an API and plugin system that is easy to use. [2] Additionally, Popcorn.js has support for playing third party media such as YouTube videos or Soundcloud audio files. [2] This could be helpful for our tool by giving instructors a way to link to instructional content that they may have already created.

Popcorn.js will be advantageous to use in our tool because of its ease of use, its plugin support, and what it can accomplish in regards to our project. Popcorn.js is simple to setup and use after downloading. In fact, the basic example on the Popcorn.js website uses only 7 lines of code to embed a video and play an annotation at a set point in the video. [3] In addition to its ease of use, Popcorn.js also has several plugins that will be helpful for our project. These plugins are, “Code” and “Pause”. The Code plugin will allow us to run code to change the DOM based upon the time in the video or audio that is playing. One example of this that will be utilized in our tool is highlighting specific lines of code as they are being discussed in the video/audio so that the student can follow along with ease. The Pause plugin is pretty self-explanatory, but still useful. It allows media to pause itself when a student clicks on a link on the page. This could be used so that the demonstration will stop playing when a student clicks to go to a reference page and can come right back to where they were.

Ultimately, Popcorn.js will be used within our project in the student-facing application as a way for students to follow along with code explanations and to also play media files related to certain points of interest within the selected piece of code the student is learning about.

***Highlight.js***

Highlight.js is a JavaScript syntax highlighter that works on both the server and the client that can automatically detect many different languages. [4] The implications this library has for our project is fairly obvious. It will be incredibly beneficial to have syntax highlighting for nearly any language that an instructor is teaching. Without the syntax highlighting the code would be very difficult to decipher.

As previously mentioned, Highlight.js can automatically detect and highlight many languages. For this auto-detection to work the code must be contained within <pre><code> tags. In case the auto-detection fails, it is also possible to include a class with the <code> tag to manually set the language type. Using Highlight.js is as simple as importing the library and calling the initialization function. This ease of use makes this library a great choice for our tool.

**Sources:**

1. <http://popcornjs.org/>
2. <http://en.wikipedia.org/wiki/Popcorn.js>
3. <http://popcornjs.org/popcorn-docs/getting-started/>
4. https://github.com/isagalaev/highlight.js

**Backend Research – Luke Turner:**

The backend services for the PIPSQUEAK project will need to provide a RESTful interface to the frontend system. There are more web service frameworks available to choose from than are reasonable to even begin evaluating, and what’s worse is nearly every available framework goes well beyond the scope of what is actually needed. The analogy about frameworks given at joelonsoftware.com (2005) about needing to build a factory to get a hammer to drive a nail still holds true despite the 9 years since it was written. With that in mind, we looked into a handful of modern frameworks with the knowledge that we were going to be relying on a JavaScript frontend and that we only need a handful of very specific RESTful features.

With our team’s strong emphasis on JavaScript skills, the most obvious full-fledged backend framework is Node.js. However, if we decide to go with a SQL based database, we also have the options to use Amazon’s AWS or Microsoft’s Azure based built in database API’s. Alternatively we could just go full bore and write our own Java based backend from scratch, which, depending on the complexity of our final design could be a boon to us down the road. We will analyze each of these options and base our final backend design decision on the following analysis of each option.

A full-size Java based backend that we design from the ground up would certainlyprovide us with the most firepower for what we are aiming to do with our software. There are some very specific features that we would like to implement on the backend and many full featured frameworks would carry a lot of deadweight. There are also many powerful tools that would be available to us using this option, such as integrating with MongoDB, using a Tomcat server, or Vert.X to increase scalability. The main drawback with this approach is a large tax on independent learning for most of our group to become familiar with the tools, techniques, and testing frameworks. On top of the learning curve, there is also the cost and maintenance increase with this option with it requiring a full featured virtual server to run in.

Amazon AWS would provide us with the RDS and DynamoDB options which provide full REST API’s for relational and NoSQL based databases without the need to write a full backend. The APIs available cover most web languages, and while less flexible in tailoring functionality, provide 99.99% availability and the full weight of Amazon for service and built in features. The main draw to this approach is we could quickly read the API documentation, find the best fit for our project, and move on to focusing on the front end, without needing to divide our time committing to two separate but intertwined pieces of our software. The main drawback here, again, is a lack of customization to our specific problem, guaranteed cost now and in the future, and by marrying our product to Amazon’s API’s we would be essentially limiting ourselves and any future users to that single vendor for service.

Microsoft’s Azure services have many of the same pros and cons associated with Amazon AWS (with a few exceptions). With relational and NoSQL options available here as well and full featured API’s for most web languages, they parallel quite closely with the options seen with Amazon’s RDS and Dynamo but with a distinct feature that makes it worth looking at. Unlike Amazon’s current RDS and Dynamo implementations, Microsoft’s Azure DocumentDB and SQL Database options provide optional JavaScript and Visual Basic API endpoint customization. This would allow us an additional degree of freedom when developing our tools to not be forced into using any one tool a specific way. With an additional boon comes and additional bane, in that Azure services are generally a little more expensive than equivalent services found on Amazon. This drawback weighs even more heavily on the fact that utilization of Azure’s API would likewise etch our software to using Microsoft going forward, limiting the options for us and our users down the road.

Finally we look to Node.js, which, it should be said, could be used with all three of the above options as well if we wanted to use features from it. It can, however, also stand on its own, and is an interesting evolution for web development in its own right. Node.js provides functionality to build a REST API connected to MongoDB (using Node Driver and/or Mongoose), as well as most SQL based options. Using Node.js would eliminate much of the learning curve associated with the Java option, while still allowing a lot of the flexibility we saw with customizing our own toolset. Similar to MongoDB, Node.js is event driven by nature and could be scaled horizontally (using clusters) if there is sufficient need. This is worth mentioning simply because due to JavaScript’s asymmetric nature, this didn’t used to be the case and was one of the main drawbacks to this approach. The remaining drawbacks to Node.js include terrible debugging time (no static type checking, generally poor indicators of what exactly went wrong and where), file manipulation can be tricky which is very relevant to our project, and the documentation and available libraries can be challenging to navigate and is scattered in terms of functionality and usefulness.

To summarize these findings we see that both the Java and Node.js based backends would lend heavily to allowing us to customize the backend to our specific needs, while the Amazon AWS and Microsoft Azure would allow us the luxury of turning most of our attention to the frontend and leaving the backend in the API platform we choose. Database choice will heavily influence these decisions as well, since Node.js will gain a lot more weight if we choose MongoDB, and the Azure and AWS options will come in much higher if we choose PostgreSQL or MySQL. Java could swing any direction in regards to database options as there are libraries to account for nearly every one of them.

**Sources:**

<http://discuss.joelonsoftware.com/>

<http://meteor.com/>

<http://nodejs.org/>

<http://docs.oracle.com/>

<http://aws.amazon.com/>

<http://azure.microsoft.com/en-us/>

**Database Research – Scott Gavin:**

When looking at the different databases out there the question must be asked: “What is my intended use?” For this project we are intending to use RESTful services, so we want a technology that will work well with that. The traditional model for databases can work just fine with rest, but these usually require PHP or some kind of server-side middleman language. This isn’t really necessary, as there are some models that allow the client to talk to the directly. For this projects needs we’ll want to store web pages, or locations of web pages, that contain a professors code and video/audio for instruction purposes. We’ll also need to store user data, specifically for the content creators to ensure they are the only ones who can create or edit their content under their name. With these parameters in mind we’ll be looking at a few different technologies: PostgreSQL, MySQL, MongoDB, CouchDB, and Redis. There are other options available, but these are what I chose to look at. Hadoop would be completely unnecessary for our purposes, and would run slower for data such as this than a different database. SQLite could work, but it’s essentially a hamstrung version of MySQL, so again it wouldn’t be a good fit for this project.

PostgreSQL is the first DBMS any student attending the University of Missouri will encounter. It uses the relational database model is very useful for maintaining reliability and consistency of data. It’s probably also the database most of the group members are familiar with. As far as speed of queries and storage goes it is pretty comparable with MySQL. It must use a server-side manipulation language such as PHP to access the data and serve it to the user.

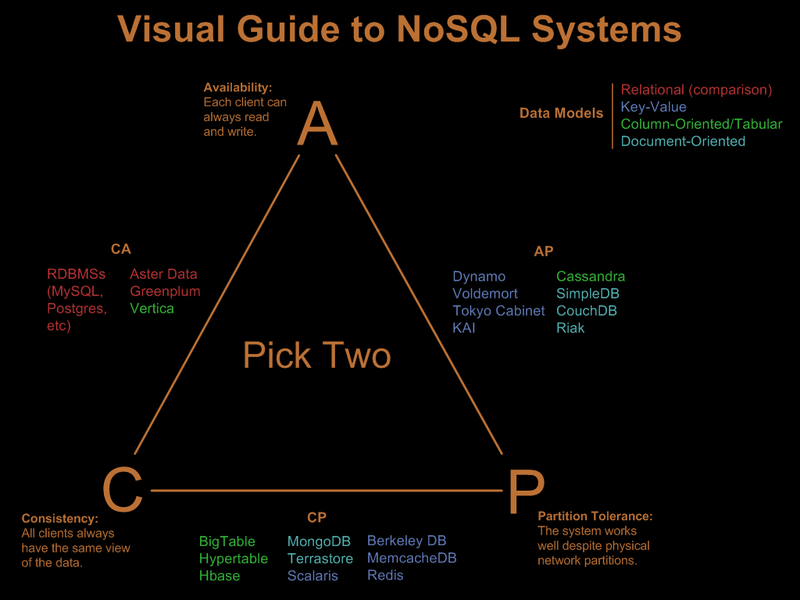
MySQL is equivalent to PostgreSQL in almost all ways. It is easier to read and use as it uses logical words as the keywords and query words, but it is also a little more difficult to use with PHP. As with PostgreSQL is requires a middleman to serve its contents to a user. When compared to PostgreSQL MySQL uses a more standard syntax of SQL, with PostgreSQL modifying some of the language. MySQL would be the better choice if the team decides to store files in the database in binary. If that is not the case however, it is basically equal to PostgreSQL, with the differences being more a matter of personal taste than any real differences.

MongoDB is one of the most popular databases in the NoSQL movement. It is based on JSON and supports easy scalability while still maintaining a familiar query structure for those coming from MySQL. It also has several REST interfaces to choose from to support the application using RESTful methodologies. It has quite a bit of flexibility and runs faster than MySQL; however it is something of a disk hog. MySQL can also support client to database connections without requiring an intermediary.

CouchDB is another popular NoSQL database, and it brags that it was designed of the web, meaning it works very well with web technologies. It is built to natively support rest, and as with MongoDB it scales fairly well, and has easy backups. However, it is very strict with its query structure. It is designed for content that is to be put into the database and then edited rarely. This means that, for the purposes of the project, if a professor created a page and wanted to edit it, it would be trickier and perhaps less efficient than with another database. It also allows the client to connect directly to the database, which would allow faster queries and stores than with requiring the intermediary.

Redis is another option that is known to be immensely fast. It is designed for real time analysis and for databases that will have a known size. Redis also needs an intermediary, and does have PHP client supports.

In conclusion there are a number of options. MySQL would probably be the more ideal option for the relational databases, as it is slightly easier to use. MongoDB or CouchDB are also excellent options, and they could both be used very effectively and quickly, but Mongo may be the better choice because of its flexibility and relative security. The biggest factor between MongoDB and CouchDB that sways things either way is that MongoDB is consistent, while CouchDB is instead available as can be seen in the image below, obtained from <http://blog.nahurst.com/visual-guide-to-nosql-systems>. Redis could work, but its intended use is a little beyond what well need it for, and we don’t really need the real-time analytics. We also don’t want to try and have a known amount of data; the content should be able to keep expanding.



**Sources:**

<http://blog.nahurst.com/visual-guide-to-nosql-systems>

<http://couchdb.apache.org/>

<http://redis.io/>

<http://www.mongodb.org/>

<http://www.scottlogic.com/blog/2014/08/04/mongodb-vs-couchdb.html>

**Bios:**

**Andrew Dominick:**

Andrew Dominick is a senior at the University of Missouri. He is pursuing a Bachelor’s Degree in Information Technology. Andrew is from Lee’s Summit, Missouri. He will fill the role of product owner and also work as a developer on this project. Since beginning his studies at the university, Andrew has gained a wealth of knowledge in various technologies and programming languages. Andrew has experience programming in C, Objective C, C#, Visual Basic, HTML, CSS, PHP, JavaScript, and SQL. He has also recently begun working with Java and Ruby on Rails. Andrew has contributed to several other team software projects in the past, including an iPad application for on-scene firefighters and an assignment submission program to be used by college students. Currently, Andrew is interning at the Tiger Institute for Health Innovation, where he helps develop innovative healthcare technology and software. When he is not programming, Andrew enjoys playing tennis, video games, cheering on his favorite sports teams, and enjoying the outdoors.

**Scott Gavin:**

Scott is a senior working towards a bachelor’s degree of the sciences in Computer Science at the University of Missouri in Columbia. He has a plethora of experience using web technologies and databases to create solutions for given problems. He also has a passion for exploring new technologies and techniques for creating content. He was born in Washington State to a military family, and as such has lived many different places in America, finally settling in Missouri.

He currently primarily uses different web platforms and APIs to design and develop resources, but he is very interested in Java programming and has been teaching himself Java gradually. As with any computer science student at the University of Missouri, he also has an extensive background using C. In addition, he has a good deal of experience with Databases in general and SQL in particular, and he has been exploring the different databases that are available beyond the standard relational models.

**William Moors:**

Bill is currently a senior planning to graduate in the spring of 2015 with a Bachelor of Science in Information Technology (BSIT). In terms of information technology, Bill is mostly interested in web development, although database design and UNIX administration have also caught his attention. In addition to information technology, Bill has maintained an interest in science and has a goal of attending an optometry program after he graduates with his BSIT.

As far as experience goes, Bill worked in the information technology field in the Marine Corps before attending college. During his time in the Marines, he was mostly responsible for basic software and hardware maintenance of computers, networks, and servers. Although this did not include much programming, it was valuable experience that has helped him stay in the top ten percent of his class as an undergraduate.

From his recent education, Bill also has experience programming in C, HTML, PHP, CSS, JavaScript, AJAX, and PostgreSQL. Although he is capable of programming in all of these languages, Bill prefers to focus on frontend programming with HTML, CSS, and JavaScript. Bill has been involved in two other team projects including a ski resort website and a file submission program. Bill also has experience with Git version control for group collaboration.

**Luke Turner:**

Luke Turner is a senior in Computer Science and an Air Force veteran with a degree in Korean. He mainly translated Korean while in the Air Force, but he did a lot of Java and JavaScript programming on the side. Luke is from St. Louis originally and has lived in Monterey (California), Seoul (South Korea), and Washington D.C. prior to moving here to Columbia. After getting out of the Air Force, he started using the GI Bill to attend Mizzou while running his own IT consulting company geared primarily toward supporting non-profit companies. Currently Luke is a software developer at Shelter Insurance writing backend services for a web application on an Agile Scrum team.

Luke's programing experience includes familiarity with C, C++, Maya (3d modeling), HTML, and some Python. He is adept at Scala, AngularJS (Javascript), SQL, Cassandra, MongoDB, and PHP, and an expert in Java, Groovy, and Spock (the unit testing platform). While he is capable of building frontend systems, he prefers to direct his focus on building backend systems such as REST API’s, NoSQL databases on Cassandra/MongoDB, or writing unit tests.

**Avery Wells:**

Avery Wells was born and raised in Joplin, Missouri and attended Webb City High School. This is Avery’s fourth and final year at the University of Missouri, and in May he will be graduating with a Bachelor’s of Science in Computer Science. After graduation, Avery plans to continue his education and obtain a Master’s of Science in Computer Science with an emphasis in bioinformatics from the University of Missouri. Outside of the classroom, Avery enjoys golfing, watching and attending Mizzou sporting events, playing guitar, and working on personal software projects.

On the technical side, Avery is most proficient in web technologies, especially JavaScript and jQuery. During the summer of 2014, Avery interned with DST Systems, where he used JavaScript extensively to write integration tests for an online workflow management system that DST produced. Avery also worked with JavaScript and other web technologies during the spring 2014 semester for two class projects. Both projects were web based games that relied on JavaScript for the majority of the game logic and execution.

In addition to JavaScript, Avery is familiar with C, Java, PHP, and Python. Avery has used these programming languages throughout his coursework and has used them in several projects.

**Progress Reports:**

**Paper Report:**

For the writing of our paper, we decided to practice Agile Scrum and the new planning and work board tools we have, using Scrum methods for developing our paper. We split the paper into several parts and split those parts into sub-tasks, which we divided each week (sprint) to write our paper in an extremely fast manner.

Sprint 0:

For Sprint 0 we created our JIRA board (workboard tool) and wrote the paper stories we needed for the upcoming sprints (created a backlog).

Sprint 1:

For Sprint 1 we created our testing plan, selected our individual research topics, created our paper’s title page, and wrote our individual biographies.

Sprint 2:

In Sprint 2 we wrote the testing plan out in our paper, as well as writing various analysis documents (cost analysis, and code analysis). Finally, we loaded all existing documents into a single master paper document hosted on Microsoft’s OneDrive.

Sprint 3:

Sprint 3 was a fairly large sprint and included the written introduction for our paper, creating a change log for the paper document as well as a section for when we start coding, and the written portion of our individual research papers. We also wrote out our technical requirements, created our standards and identified constraints, wrote the project management plan, and created sections for security and the overall objective of our project.

Sprint 4:

For Sprint 4 we had several tasks with the goal of completing as much as possible for our rough draft of the design document. We created the Table of Contents, added in Results, Discussion, and Progress report sections, and completed the design for our project. We also wrote our testing plan and created mock design images for the applications two main views.

Sprint 5:

For Sprint 5 we combined all of the remaining documents into a single document. We then each read through the whole paper and proofread it while making corrections. We each then added in relevant graphics such as UML and database design charts. Finally, we synthesized the paper into one voice and did a final proofreading.

**Development:**

We continued to practice Agile Scrum for the development portion of our project. The development was split up into several milestones to be completed throughout the semester.

Sprint 1:

For the first sprint we regrouped and got back in communication as a group. We also did a basic template for the instructor view and basic file system set up. Lastly, we merged our fronted into the NW.js framework instead of the web app we had planned on.

Sprint 2:

During the second sprint we added buttons for all of the manipulation functions and the code and video upload. The functionality for these buttons on the backend side was also completed. The history and list of events was finished as well.

Sprint 3:

In this sprint we moved from the instructor view to focusing on the student view. We generated the Popcorn.js file the the student view would run and got it to properly fire on the video playback events.

Sprint 4:

For our last sprint we focused heavily on testing and fixing bugs. We also made final additions to our documentation and put together our final presentation.

**Licenses:**

**AngularJS License:**

MIT License

Source: <https://github.com/angular/angular.js/blob/master/LICENSE>

**Node.js License:**

MIT License

Source: <https://raw.githubusercontent.com/joyent/node/v0.10.33/LICENSE>

Contributor License

Source: <http://nodejs.org/cla.html>

**MongoDB Licenses:**

GNU Affero General Public License

Source: <http://www.gnu.org/licenses/agpl-3.0.html>

Apache License v2.0

Source: <http://www.apache.org/licenses/LICENSE-2.0>

Commercial licenses may also be obtained by contacting the company via this link: <http://www.mongodb.com/lp/contact?_ga=1.87944216.1220895523.1415764086>

**Popcorn.js**

MIT License

Source: <http://popcornjs.org/code/LICENSE>

**Highlight.js**

MIT License

Source: <https://github.com/isagalaev/highlight.js/blob/master/LICENSE>

**CodeMirror**

MIT License

Source: <http://codemirror.net/LICENSE>

**Bootstrap**

MIT License

Source: <http://getbootstrap.com/getting-started/>

**FAQ:**

* How do I access the lecture I just created?
  + The new lecture will be saved to the directory you specified in the “index.html” file. It will also open up in a new NW.js window.
* What happens if I want to undo an action?
  + You can click on “view” at the top right of the application to access the history of actions. Click on the “x” to remove one.
* How do I use this application?
  + Select one or multiple lines of code when the video is at the time you want the manipulation to start. Press the button for the function you want to act on the lines you selected. Enter the duration you want the function to last. Repeat as necessary for other functions.

**Additional References:**

1. Node.js by Joyent October 28, 2014.

<http://nodejs.org/api/>

1. MongoDB November 3, 2014

<http://www.mongodb.org/>

1. Angular.js by Google November 4, 2014

<https://angularjs.org/>

1. Node-Style-Guide by Felixge November 2, 2014

<https://github.com/felixge/node-style-guide>

1. Building A Node.JS Server That Won’t Melt by Mozilla January 15, 2013

<https://hacks.mozilla.org/2013/01/building-a-node-js-server-that-wont-melt-a-node-js-holiday-season-part-5/>

**Changelog (paper):**

**V0.1**

Administrative:

Implemented JIRA as sprint management

Design:

No design issues addressed

Implementation:

Added Biographies sections

Added Title Page

**V0.3**

Administrative:

Initiated testing and analysis plan

Design:

No design issues addressed

Implementation:

Added Cost Analysis Section

Merged Previous Documents into one

**V0.5**

Administrative:

Decided on CodeMirror vs Highlight.js: CodeMirror

Design:

Settled on design for UI

Implementation:

Added Changelog

Added security section

Added research paper section

Added a project management plan section

Added introduction

Added constraints and standards

Added overall objectives section

Added design section

**V0.7**

Administrative:

Assembled all the paper components into a single document

Design:

Created several relevant graphics

Implementation:

Added Graphics

Proofread the whole paper once more

**V1.0**

Administrative:

Plan for presentation and proofread the graphics

Design:

Made the paper one voice

Implementation:

Rearranged the paper to flow better

Rewrote sections to be in one voice

**V1.1**

Administrative:

Plan for final presentation and final version of documentation

Design:

Updated current graphics to match final implementation

Implementation:

Rewrote sections to reflect final version

Added sections that were not completed last semester

**Git Commit Log:**

<https://github.com/SakoGuru/PIPSQUEAK/commits/master>